经典趣题“老鼠与毒药问题”

推 广 研 究

参赛队员　　白天衣　　项思陶

指导老师　　张　雷

参赛学校　　东北育才中学

2011 年 **8** 月

摘　要

在一次偶然的阅读中，我们发现经典的“老鼠与毒药问题”有很大的推广空间。

老鼠与毒药问题
有 1000 只一模一样的瓶子，其中 999 瓶是无毒的药物，一瓶是毒药。任何喝下毒药的生物都会在一星期之后死亡。现在，你只有 10 只老鼠和一星期的时间，如何检验出哪个瓶子里有毒药？

本文中，我们对这个美妙的组合趣题进行了三个方面的推广：
（1）对题目中的两个“一星期”作一般化时间推广，并深化了进位制解法来解决它。
（2）给“老鼠”加上一个服药瓶数的限定条件。题目更“实际化”的同时，也更增加了问题的难度。我们提出了数表-填充法，并经过可行性论证予以实现。
（3）将“毒药”数量由“一瓶”改成“两瓶”后，问题的复杂度大大增加了，为此我们设计了坐标-分组方案进行研究。在计算机的辅助下，用搜索算法编程得到了小规模情形下的精确解，用随机数算法编程对较大数据量情形也进行了快速估计。在取得重要进展的同时，也引出许多新的猜想。

经过这次课题研讨，让我们不仅收获了许多知识和经验，还深深地体会到了数学的魅力和研究的乐趣。

Abstract

When reading the problem of ‘Mice and the Poison’,we happened to notice that if we change the conditions of it,lots of interesting new conclusions will turn out.

The original problem of ‘Mice and the Poison’:
There are 1000 bottles of liquid with one of them poisoned.A mouse will die after one week if it takes the poison,otherwise it will remain living.Can you use only 10 mice to check out which bottle contains the poison in one week’s time?

In the following essay,we mainly study three generalized versions of the problem:
(1)We generalize the two ‘one week’ and develope the binary solution to the original problem.
(2)To make the problem more realistic,we add a new restriction to the ‘mice’,which also makes the problem more difficult.We create a new method called ‘Chart-Filling’ Method,and adopt feasibility demonstration of it to solve the new problem at last.
(3)Replacing ‘the poison’ with ‘two poisons’,we encounter a big challenge,which ‘Coordinate-Group’ Scheme is contrived to tackle.Computer programming is also employed to work out both exact solutions to small-scale situations by backtracking arithmetic and appraisal solutions to bigger ones by random arithmetic.Key conclusions and improvements are reached and new conjectures are raised with all these efforts.
After this wonderful peregrinate of mathemetics,we not only obtain knowledge and skills but also experience the charisma of math and enjoyment in researching.

目　　录

引　言 .... - 3 -
第一章 关于时间问题的推广 .... - 4 -
　1.1　问题的描述 .... - 4 -
　1.2　进位制解法 .... - 4 -
　　1.2.1　潜伏期t=1 的情形 .... - 4 -
　　1.2.2　潜伏期t>1 的情形 .... - 4 -
　　1.2.3　实验“独立性”和隐含的离散条件 .... - 5 -
　1.3　最优性证明 .... - 5 -
　1.4　“不死老鼠”情形 .... - 6 -
第二章 限定老鼠服药瓶数的推广 .... - 7 -
　2.1　问题的描述 .... - 7 -
　2.2　数表-填充方法 .... - 7 -
　2.3　可行性证明 .... - 9 -
　2.4　对实验轮数的改进 .... - 13 -
第三章 两瓶毒药问题的推广 .... - 14 -
　3.1　问题的描述 .... - 14 -
　3.2　坐标方案 .... - 14 -
　　3.2.1 基础方案：5×5 方案 .... - 14 -
　　3.2.2 改进方案Ⅰ：6×6 方案 .... - 15 -
　　3.2.3 改进方案Ⅱ：4×4×5 方案 .... - 15 -
　3.3　坐标-分组方案 .... - 15 -
　3.4　对f(n)的定性估计 .... - 16 -
　　3.4.1　二进制描述 .... - 16 -
　　3.4.2　“合成”运算和识别条件 .... - 17 -
　　3.4.3　f(n)上界 .... - 17 -
　　3.4.4　单调性 .... - 17 -
　　3.4.5　f(n)下界与一个重要的函数不等式 .... - 18 -
　　3.4.6　f(n)定义域的连续化和进一步的下界估计 .... - 18 -
　　3.4.7　f(n)下界与幂函数的比较 .... - 18 -
　3.5　对f(n)的定量计算 (计算机算法) .... - 19 -
　　3.5.1 精确解的搜索算法 .... - 19 -
　　3.5.2 快速估计的随机数算法 .... - 20 -
　3.6　对f(n)的猜想与展望 .... - 20 -
　3.7　蕴含问题 .... - 20 -
　3.8　对坐标-分组方案的新理解 .... - 21 -
　3.9　c(n)上界估计与信息论方法 .... - 22 -
总结与展望 .... - 23 -
致谢 .... - 23 -
附录A　3.5.1 节源程序 .... - 24 -
附录B　3.5.2 节源程序 .... - 25 -
附录C　n≤20 时的f(n)对应方案 .... - 26 -

引　言

在一次偶然的阅读中，来自网页http://www.matrix67.com/blog/archives/4361的这样一道经典趣味组合问题吸引了我们的好奇心：

老鼠与毒药问题

有 1000 只一模一样的瓶子，其中 999 瓶是无毒的药物，一瓶是毒药。任何喝下毒药的生物都会在一星期之后死亡。现在，你只有 10 只老鼠和一星期的时间，如何检验出哪个瓶子里有毒药？

这个看起来似乎不可能完成的问题有一个非常巧妙的解答：先把瓶子从 0 到 999 依次编号，然后全部转换为 10 位二进制数编号。让第一只老鼠喝所有二进制数右起第一位是 1 的药物，第二只老鼠喝所有二进制数右起第二位是 1 的药物，以此类推。一星期后，若第一只老鼠死了，就知道毒药瓶子的二进制编号中，右起第一位是 1，否则是 0，每只老鼠的死活都能确定出 10 位二进制数中的一位，由此便可知道毒药瓶子的编号了。

上面的解法灵活地运用了进位制思想，最大判定的数量是 10 位的二进制数，即 2^{10}=1024。

该网页的作者还对这个问题进行了一个推广：

如果你有两个星期的时间（换句话说你可以做两轮实验），为了从 1000 瓶药物中找出毒药，你最少需要几只老鼠？（第一轮实验中死掉的老鼠无法参与第二轮实验）。

同样采用类似的进位制思想，解法如下：

7 只老鼠就够了。首先，把所有瓶子从 0 到 999 编号，然后全部转换为 7 位三进制数（事实上，7 只老鼠足以从 3^7 = 2187 瓶药物中找出毒药来）。让第一只老鼠喝所有三进制数右起第一位是 2 的药物，让第二只老鼠喝所有三进制数右起第二位是 2 的药物，以此类推。一星期之后，如果第一只老鼠死了，就知道毒药瓶子的三进制编号中，右起第一位是 2；如果第二只老鼠没死，就知道毒药瓶子的三进制编号中，右起第二位不是 2，只可能是 0 或者 1。于是，问题就归约到了第二轮实验，等等。在接下来一星期里，让每只活着的老鼠继续自己未完成的任务，喝掉它负责的那一位是 1 的药物。毒药瓶子的三进制编号便全部揭晓了。

后来，该网页评论中有人提出了一个新问题：如果毒药不止一瓶，怎么安排实验才能确保检测最多瓶的药物无毒呢？不过，他没有给出详细的解决方案。

上述种种问题极大地激发了我们的研究热情，因此，我们尝试从各个角度推广并深入地思考：

（1）对题目中的两个“一星期”作一般化时间推广，并深化进位制解法来解决它。

（2）给“老鼠”加上一个服药瓶数的限定条件，使问题更“实际化”。用数表-填充法做更本质的理解，并经过可行性论证予以实现。

（3）将“毒药”数量由“一瓶”改成“两瓶”，并设计坐标-分组案进行研究。在计算机的辅助下，用搜索算法编程得到小规模情形下的精确解，用随机数算法编程对较大数据量情形进行快速估计，并给出问题的一些阶段性结论，同时引出许多新的猜想。

第一章 关于时间问题的推广

1.1 问题的描述

引言中对老鼠与毒药问题在实验时间为一周和两周时的情况进行了讨论。作为推广，下面将对一般的 m 周情况进行研究。同时注意到，引言中有一个隐含的限定，就是毒药的“潜伏期”始终为 1 周，对于这个因素，我们也将它视作变量进行一般性推广，并将新问题完整描述如下：

问题 1 当只有一瓶毒药、毒药的潜伏期为t周时，给你n只老鼠和m星期的时间，最多能确保从多少瓶药物中找出这瓶毒药($n,m,t\in\mathbf{N}_+$, $t\leqslant m$)?

为了方便描述，下面设这个问题的最优解为 $a(n,m,t)$。

1.2 进位制解法

引言中，我们利用二进制方法解决了原始问题，这里我们把它推广为一般的进位制解法来研究**问题 1**。

1.2.1 潜伏期 t=1 的情形

直接推广二进制方法，得到m周时的方案：将所有的药物编号成n位$m+1$ 进制数。若第i($i\in\mathbf{N}_+$,$1\leqslant i\leqslant n$)只老鼠在第j($j\in\mathbf{N}$,$0\leqslant j\leqslant m$，刚开始时算作$j=0$)周末死亡，则可知毒药编号的右起第i位是j；第i只老鼠第j周末未死，就让它喝掉所有编号右起第i位是$j+1$ 的药物。如果该老鼠最终生还，则该数位上的数字为 0。这样，本方案最多能确保检测的药物数量是n位$m+1$ 进制数的总数，即$(m+1)^n$ 。

所以 ，$a(n,m,1)\geqslant(m+1)^n$。

1.2.2 潜伏期 t>1 的情形

表面上看，对毒药潜伏期的推广似乎是平凡的，因为只要改变一下问题中的单位时间似乎就解决了问题，比如 $m=4$, $t=2$ 时，视两周为一个单位，则貌似与 $m=2,t=1$ 完全等价。然而，由于等待药物发作的老鼠们仍可以正常的进行实验，这将导致一只已喝下毒药而未等到其发作的老鼠又参与了下一轮实验。于是，“有效” 的实验次数是 $m-t+1$，即前 $m-t+1$ 周不管老鼠是否已喝下毒药，只要其生存都可以进行试验并得到结果，故曰“有效”;而最后 $t-1$ 周内做的任何实验在全部时间结束之前无法得到任何结果，所以是“无效的”。

因此，我们设计了“压缩”的方式来进行实验：

将所有的药物编号成 n 位 $m-t+2$ 进制数。

如果第i ($i\in\mathbf{N}_+$,$1\leqslant i\leqslant n$)只老鼠在第j($j\in\mathbf{N}$,$0\leqslant j\leqslant m$，开始时算作$j=0$) 周末死亡，则可知药

物编号的右起第i位是$j+1-t$；若第$j(j\leqslant m-t)$周末第i只老鼠未死，就让它喝掉所有编号右起第i位是j+1 的药物。如果该老鼠最终生还，则该数位上的数字为 0。

因此，$a(n,m,t)\geqslant(m-t+2)^n$。

1.2.3　实验“独立性”和隐含的离散条件

之所以能够在不知老鼠是否中毒时做下一轮实验，是因为方案中同一只老鼠所做的各次实验涉及药物的集合没有交集。假设一只老鼠被安排进行s次试验，涉及的药物集合为$P_1,P_2,\cdots,P_s$。在进位制方案中，$P_i\cap P_j=\phi$ $(\forall i,j\in\{1,2,\cdots,s\},i\neq j)$，因为毒药只有一瓶，所以老鼠对所有$s$轮实验的反馈只有至多一次是死亡。当老鼠因为$P_i$死亡（根据时间判断）时已经判断出毒药在$P_i$中、一定不属于$P_j$（$j\neq i$），因此该老鼠的其他实验结果已经确定是“没有毒药”，从而无需再等实验结果。

综上，当一只老鼠做的实验涉及药物的集合没有交集时，这些实验可以被看做“独立”实验，而无需考虑因为老鼠中毒无法正常完成后面实验的情况。

根据上文的描述，老鼠与毒药的问题有一个隐含的条件，即每周只能对每只老鼠做一次实验，否则两周的时间就可以做无穷次实验。比如说每秒做一次实验，第一周的每一秒都是“有效的”，用进位制方法能使之满足同一只老鼠每次实验用药物的集合没有交集，因此每秒能做一轮独立的实验。同样，无限细化时间间隔则导致无穷。因此，这个离散化条件是必要的，下面的所有论证将以此为基础。

从这里我们可以看出，真正关键的因素是实验轮数，而时间变量 m 和 t 可以看作是通过影响实验轮数 $m-t+1$ 来间接影响实验结果的。

1.3　最优性证明

虽然 1.2 的进位制方案简洁美妙，但是**问题 1** 中要求的是“最优”方案，而进位制方案的最优性还没有得到保证。因此，我们将证明以下定理。

定理 1　$a(n,m,t)=(m-t+2)^n$

证明：把老鼠简化成一个变量，并考虑其可能的状态。因为每只老鼠的死亡时间可以是第t周末到第m周末，所以有$m-(t-1)$种死亡状态和一种生还状态，共$m-t+2$ 种状态。于是由乘法原理，所有老鼠的生死状态共$(m-t+2)^n$种。

而可能的毒药状态数为$C^1_{a(n,m,t)}=a(n,m,t)$（每瓶都可能是毒药）。无论用什么方案，无论毒药的分布如何，我们都将得到$(m-t+2)^n$种可能的老鼠生死状态之一，并需据此判断出毒药是哪一瓶。因此，假设毒药的状态数$a(n,m,t)$多于老鼠的生死状态数$(m-t+2)^n$，由抽屉原理必有至少两种毒药的状态对应于同一种老鼠生死状态，那么，得到这种老鼠的生死状态时将无法区分这两种情况，从而无法判断哪瓶是毒药，这与问题.1 矛盾。

所以　$a(n,m,t)\leqslant(m-t+2)^n$ 。

结合 1.2.2 节的$a(n,m,t)\geqslant(m-t+2)^n$，即得$a(n,m,t)=(m-t+2)^n$。证毕。

这样，我们就得到了函数 a 的确切表达，并证明了进位制方案的最优性。

1.4 “不死老鼠”情形

1.2.3 节中讨论了试验中的老鼠中毒死亡造成的影响，这自然引出一个新的设想：假如毒药发作时，老鼠只是发出异常叫声而并不死亡，即“不死老鼠”，不影响进行下一轮试验，那么会对结果有什么影响呢？

为此，我们提出下面的问题：

对n只老鼠做m轮实验且老鼠不因服食毒药死亡而是仅仅在毒药发作时发出异常叫声时，最多能确保从多少瓶药物中找出一瓶毒药？($n,m\in\mathbf{N}_+$) (时间的问题已经在上文中完全解决，为方便描述，此处用实验轮数代替时间)

设这个问题的最优解为$a^*(n,m)$。

既然是“不死老鼠”，那么类似 1.3 节的最优性论证给出的最大值就将发生改变。这时，相应的“生死状态”将变成“叫声状态”。“叫声状态”分为正常或异常两种状态。每次实验每只老鼠将有 2^m种可能状态，故总的“叫声状态”数为 2^{mn}。所以$a^*(n,m)\leqslant 2^{mn}$。

在上文最优性证明所用方法的启发下，不难给出达到$a^*(n,m)=2^{mn}$的方案：

用$m\times n$的 0/1 方阵为 2^{mn}瓶药编号，并用$a_{i,\,jk}$表示第i号方阵第j行第k列上的数字。在第j轮实验初让第k只老鼠喝下所有使得$a_{i,\,jk}=1$ 的药物i，由第j轮实验末第k只老鼠的反应即可判断毒药方阵该位置的数字。

第二章 限定老鼠服药瓶数的推广

2.1 问题的描述

在引言的进位制方案中，让一只老鼠一次性服食几百瓶药显然不现实。由此联想到一种改进方式：限定每只老鼠每轮实验服用药物的瓶数。现将新问题完整描述如下：

问题 2 用n只老鼠进行一轮实验，其中每只老鼠至多服用r瓶药时，最多能确保从多少瓶药中选出一瓶毒药？($n,r \in \mathbf{N}_+$, $r \leqslant 2^{n-1}$)

这里设**问题 2** 的最优解为 $b(n,r)$。

2.2 数表-填充方法

我们将**问题 2** 转化为一个向表格内填数字的问题，命名为数表-填充法。
为了方便描述先将基本概念陈述如下：

定义 2.1 n-鼠群：有一定的服用药物数量限制的 n 只老鼠组成的群体称为一个 n-鼠群。特别的，一只已经不能再喝药的老鼠仍可以作为 n-鼠群的一个合理成员。

下文中，将把鼠群列成数表格式，每行代表一只老鼠。假如已经计划让某些老鼠服用了一些药物，则把它们视作只能吃相应的更少药物的老鼠。这样，方案建立的过程可以看作鼠群“衰减”的过程。

定义 2.2 空间：一个鼠群中某一只老鼠至多服用 s 瓶药物，则称它有 s 个空间。每只老鼠的空间数之和称为整个鼠群的空间数。一瓶药被计划给某只老鼠服用，则称它占用了一个空间。

这样，一瓶药至多占用同一只老鼠的一个空间，且所有药占用的空间总数不超过整个鼠群的空间数。

定义 2.3 p-反馈形式：一瓶药共占用了整个鼠群的 p 个空间，则称之为一个 p-反馈形式。

这里之所以称之为“反馈”形式，是因为当这瓶药有毒时，得到的“反馈”是有 p 只老鼠死亡。
服药量的限制就是鼠群空间数量的限制，即对反馈形式占用空间数量上限的限制。要达到“能够区分所有的药物”的目标就是要使得“当任意一瓶药物是毒药时，实验的结果不同”，即每瓶药的反馈形式不同。因此问题化为对一个特定的鼠群，如何用尽可能多的不同反馈形式填充它，使得没有老鼠被占用的空间数超过原有空间数。

以下用两个特例来具体介绍数表-填充方法。

（1）考虑 $n=r=4$ 的情况，我们从节约空间的角度来构造反馈形式。若药物 1 的反馈形式是‘生生生生’，即四种老鼠全生还，则药物 1 不需要老鼠服用、不占用空间，是最节约空间的模式。同样的，只需要一只老鼠参与的反馈形式共 4 种，需要两只的则有 C_4^2 种，以此类推。这里，构造一个数表描述这些反馈形式，其中，第 i 行表示第 i 只老鼠所服药物序号。整个鼠群为一个 4×4 的数表，共 16 个空间。

首先，药物 1 无须任何老鼠服用，不占空间，即为 0-反馈形式；接下来，药物 2,3,4,5 分别由 4 只老鼠服用，为 1-反馈形式，共占 4 个空间，对这五瓶药物的设计方案如下图所示。

1 号老鼠	2			
2 号老鼠	3			
3 号老鼠	4			
4 号老鼠	5			

之后，则每瓶药至少为 2-反馈形式，否则将与 1,2,3,4,5 之一重复。这时，共 C_4^2=6 种，其中每个占 2 个空间。

一共需 0×1+1×4+2×6=16 个空间，恰好与总空间数相同！

因此这个鼠群最多容纳 1+4+6=11 瓶药，即 $b(4,4)\leqslant 11$。如图所示，我们可以构造出达到这个状态的例子：

1 号老鼠	2	6	8	10
2 号老鼠	3	6	9	11
3 号老鼠	4	7	8	11
4 号老鼠	5	7	9	10

所以，答案是 $b(4,4)=11$。

（2）考虑 $n=5$，$r=10$ 的情况，同样的，把鼠群看做一个 5×10 的数表，共 50 个空间。

首先，药物 1 无须任何老鼠服用，不占空间，即为 0-反馈形式；

接下来，药物 2,3,4,5,6 分别由 5 只老鼠服用，为 1-反馈形式，共占 5 个空间；

1 号老鼠	2									
2 号老鼠	3									
3 号老鼠	4									
4 号老鼠	5									
5 号老鼠	6									

之后，每瓶药至少为 2-反馈形式，这时共 C_5^2=10 种，其中每个占 2 个空间；

1 号老鼠	2	7	8	9	10					
2 号老鼠	3	7	11	12	13					
3 号老鼠	4	14	8	11	15					
4 号老鼠	5	14	16	9	12					
5 号老鼠	6	15	16	13	10					

再之后，每瓶药至少为 3-反馈形式，这时共有 C_5^3=10 种，其中每个占 3 个空间，一共需 0×1+1×5+2×10+3×10=55 个空间，超过总空间数！因此不能填入全部的 3-反馈形式，而只

能使用 $[\frac{50-(0\times1+1\times5+2\times10)}{3}]=8$ 个 3-反馈形式（[x]表示不超过 x 的最大整数，下同）。

因此，这个鼠群最多容纳 1+5+10+8=24 瓶药，即 $b(5,10)\leqslant24$。如图所示，我们可以构造出找到达到这个值的例子。

1 号老鼠	2	7	8	9	10	17	18	19	20	21
2 号老鼠	3	7	11	12	13	17	18	22	23	24
3 号老鼠	4	14	8	11	15	17	20	24	22	21
4 号老鼠	5	14	16	9	12	18	19	20	22	23
5 号老鼠	6	15	16	13	10	19	21	24	23	

所以，答案是 $b(5,10)=24$。

数表-填充方法具有一般性，并可得到一个优秀上界，因此，下面将研究其一般性的描述：

$b(n,r)$（$r\leqslant2^{n-1}$）对应的n-鼠群共确定nr个空间，依次用所需空间数为 $0,1,2,\cdots,n$的药物对其进行填充。为了最多的填充，我们尽量先填充占用空间数少的反馈形式。令$S_i=\sum_{j=0}^{i}(C_n^j\times j)$，其中$C_n^j$为$j$-反馈形式的个数，$jC_n^j$即为全部$j$-反馈形式所占的空间数，$S_i$表示所有 $0,1,\cdots,i$-反馈形式占用的总空间数。

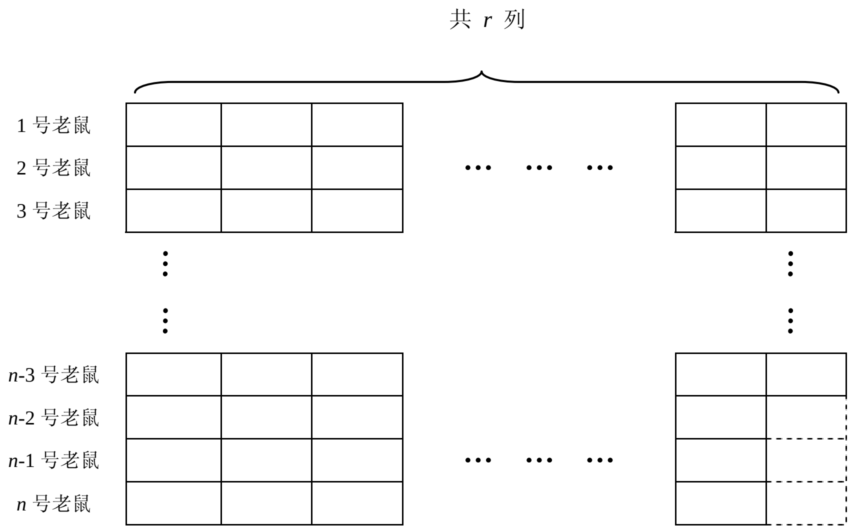
设 $i_0\in\mathbf{N}_+$ 满足 $S_{i_0-1}<nr\leqslant S_{i_0}$，则类似$b(5,10)$的例子，这$nr$个空间可以容纳至多 $[\frac{nr-S_{i_0-1}}{i_0}]+\sum_{j=0}^{i_0-1}C_n^j$ 种药物，也就证明了上界 $b(n,r)\leqslant[\frac{nr-S_{i_0-1}}{i_0}]+\sum_{j=0}^{i_0-1}C_n^j$。特别的，注意到所有反馈形式共占用 $S_n=n2^{n-1}$个空间，所以，当$r>2^{n-1}$时，填入全部反馈形式后仍有大量空间剩余。这时的情况其实就是进位制（原始问题）的情况，即此时相当于没有r的限制。因此上文中才限定$r\leqslant2^{n-1}$，以便在陈述和论证中不考虑这种特殊情况，以下也均假定$r\leqslant2^{n-1}$。

2.3 可行性证明

上节讨论中，我们得到了数表-填充方法所能达到的上界。不过，在 n、r 均很大时，是否一定有办法能达到这种理想状态呢（比如在上节 $b(5,10)$的例子构造中，如果填充得不够好，就可能剩下 4 个或更多空间，而无法再填入 3-反馈形式）？为此必须对鼠群和获取反馈形式的方案做进一步的研究。

首先，我们需要找到一种对鼠群的更深刻的描述：

定义 2.4 (n, r)-密集形式：一个 n-鼠群称为(n,r)-密集形式当且仅当所有老鼠中，每只拥有的空间数最多为 r、最少不少于 r-1。n-密集形式是 r 任意时(n,r)-密集形式的缺省表达。特别的，如果所有老鼠拥有的空间数都是 r，该鼠群称为一个(n,r)-完美密集形式。



对每组(n,r)，(n,r)-完美密集形式是唯一的。$b(n,r)$问题对应的 $n\times r$ 方阵本身就是一个(n,r)-完美密集形式，密集形式这个概念正是把方阵进行的一个推广（上图所示的例子中，前 n-3 只老鼠各有 r 个空间，后三只老鼠各有 r-1 个空间，如右下角的虚线部分所示）。一个(n,r)-密集形式所占的空间数介于$(n,r-1)$-完美密集形式和(n,r)-完美密集形式所占空间数之间。

定义 2.5　p-充分填充：一个 n-鼠群被用 p-反馈形式按某种方案填充后$(n\geqslant p)$，如果剩余未使用的空间数不足 p，即在理论上已经无法更好的填充，则称这个鼠群被 p-充分填充。特殊定义一个鼠群被0-充分填充当且仅当它被填入全部0,1,⋯,$(p-1)$-反馈形式后，被p-充分填充。

定义 2.6　特征数 Z(n, r)：对于一个(n,r)-完美密集形式（$r\geqslant 1$），一定存在正整数 i 使 $S_{i-1}<nr\leqslant S_i$（$S_i=\sum_{j=0}^{i}(C_n^j\times j)$），这个数$i$称为该$(n,r)$-完美密集形式的特征数，记作$Z(n,r)$。

这样，我们需要证明的数表-填充方法能否达到上界的问题就化为了(n,r)-完美密集形式能否被 0-充分填充的问题。

为此，先证以下三个引理：

引理 1　对于每行空间数不少于C_{n-1}^{i-1}的 n-鼠群，全部 i-反馈形式恰能填充其中每行的C_{n-1}^{i-1}个空间。（一行代表一只老鼠，下同）

证明：考虑所有含有某行分量的 i-反馈形式，它们除了该行的分量外，其他 i-1 个分量无约束地分布在 n-1 个其他的行，因此这样的反馈形式共有C_{n-1}^{i-1}个。这些反馈形式每个占据该行的一个空间，因此共占据其中的C_{n-1}^{i-1}个空间。上述论证对每行均成立，即每行都恰被占据了C_{n-1}^{i-1}个空间，证毕。

事实上，**引理 1** 保证了全部 i-反馈形式恰能占据一个(n,C_{n-1}^{i-1})-完美密集形式。因此，一个足够大的(n,r)-（完美）密集形式被填入全部 i-反馈形式后，剩下的鼠群仍是（完美）密集形式的。所以要证明(n,r)-完美密集形式能被 0-充分填充，只需证明它被填入全部

1,2,…,(*Z*(*n*,*r*)-1)-反馈形式后所剩的 (*n*,*r*- $\sum_{i=1}^{Z(n,r)-1} C_{n-1}^{i-1}$)-完美密集形式能被 *Z*(*n*,*r*)-充分填充。

引理 2 $xC_n^x=nC_{n-1}^{x-1}$

证明：构造一个充分大的 *n*-鼠群并填入全部 *x*-反馈形式。考虑用以下两种方式计算 *x*-反馈形式所占的总空间数：

1.由**引理 1**，鼠群共 *n* 行，每行被占据了 C_{n-1}^{x-1} 个空间，所以，一共有 nC_{n-1}^{x-1} 个空间被占据，这正是所有 *x*-反馈形式所占据的空间数。

2.直接考虑反馈形式的分量, 得到全体 *x*-反馈形式共有 C_n^x 个，它们每个占据 *x* 个空间，所以，一共是 xC_n^x 个空间。

上述两种方法得到的数值应该相同，所以 $xC_n^x=nC_{n-1}^{x-1}$，证毕。

引理 3　任给定一个(n, r)-密集形式，一定存在一种方法把它分成两个 n-密集形式，其中一个是任给定的(n, k)-密集形式(r ≥k 且待分密集形式的空间数不少于分得的密集形式)。

证明：设给定的(*n*,*r*)-密集形式中，有*r*个空间的老鼠共*x*只，把它们记为A_1；有*r*-1 个空间的共*n*-*x*只，记为A_2。类似的，设给定的(*n*,*k*)-密集形式中有*k*个空间的老鼠共*y*只，有*k*-1 个空间的共*n*-*y*只。

当*r*>*k*时，如果*x*≥*y*，则在A_1中取*y*只老鼠，每只拿出*k*个空间，剩下的*n*-*y*只老鼠每只拿出*k*-1 个空间，组成一个要求的(*n*,*k*)-密集形式。余下空间构成的鼠群中有*x*-*y*只（原来属于A_1，但只取了*k*-1 个空间的）拥有*r*-*k*+1 个空间，其余的有*r*-*k*个空间，这些恰构成一个*n*-密集形式。

r>*k*时，如果*x*<*y*，在A_2中取*y*-*x*只老鼠，连同A_1中的*x*只老鼠，每只拿出*k*个空间，剩下的*n*-*y*只老鼠每只拿出*k*-1 个空间，这组成一个要求的(*n*,*k*)-密集形式，同上，可以算出余下的鼠群为一个(*n*, *r*-*k*)-密集形式。

当 *r*=*k* 时，因为待分密集形式的空间数不少于分得的密集形式，所以必有 *x*≥*y*，因此直接用上面 *x*≥*y* 情况的方法即可。

综上，无论 *r* 与 *k* 大小关系如何，都存在这样一种分法达到题目的要求，证毕。

定理 2　对 i, n, r, x∈$\mathbf{N}_+$，一个共有x个空间的(n, r)-密集形式只要满足 1≤i≤n, x≤i C_n^i，就能被i-反馈形式充分填充。（实际上，上面的两个条件只是保证了存在能填充n-密集形式的这种反馈形式，并且它们的数量足够充分填充原密集形式）

证明：对 *n* 使用归纳法。

1)当 *n*=1 时，$\because 1\le i\le n=1 \quad \therefore i=1$

$\therefore x\le iC_n^i=1 \quad \therefore x=1$

即此时只有 1 个空间，当然能被 1-反馈形式充分填充，得证。

2)假设 *n*=*k* 时（*k*≥1）命题成立。

当 *n*=*k*+1 时，分六步证明：

Ⅰ. 由**定义 2.4** 的推论，得 *n*(*r*-1)<*x*≤*nr* **……(1)**

故 i=1 时，$x \leqslant iC_n^i = n$ ，

所以 r=1。

因此，只可能 x 只老鼠有 1 个空间而其他老鼠没有空间, 依次填充即可。

当 i=n 时，同理只可能是 $x \leqslant n$, r=1：当 $x<n$ 时无需填充；当 $x=n$ 时填入唯一一个 i-反馈形式即可。

下面只考虑 $2 \leqslant i \leqslant n-1$ 情况。

Ⅱ． 由**(1)**和题目条件，$n(r-1)<x \leqslant iC_n^i$，

又由**引理 2**，得 $n(r-1)< iC_n^i = nC_{n-1}^{i-1} \quad \therefore r-1 < C_{n-1}^{i-1}$

因为两边都是整数，所以 $r \leqslant C_{n-1}^{i-1}$ 。 ……**(2)**

又由**(1)**, 得 $r \geqslant \dfrac{x}{n}$ ……**(3)**

Ⅲ． 设 $r(i-1) \equiv y(mod\ n-1)(1 \leqslant y \leqslant n-1)$，故 $n|r(i-1)-y$ 。

从原(n,r)-密集形式中去掉含r个空间的一行（设它为X_1）后，得到一个$(n-1)$-密集形式A，A的空间数为$x-r$。

再考虑一个$(n-1, \dfrac{r(i-1)-y}{n-1}+1)$-密集形式 B, 其中 y 行有$\dfrac{r(i-1)-y}{n-1}+1$个空间。

B 的总空间数为 $y(\dfrac{r(i-1)-y}{n-1}+1)+(n-1-y)\dfrac{r(i-1)-y}{n-1}=r(i-1)$。

$\because i \leqslant n-1 \qquad \therefore \dfrac{r(i-1)-y}{n-1} < \dfrac{r(i-1)}{n-1} < r$

又 $\dfrac{r(i-1)-y}{n-1} \in \boldsymbol{N}_+$， $\therefore \dfrac{r(i-1)-y}{n-1} \leqslant r-1$，即 $\dfrac{r(i-1)-y}{n-1}+1 \leqslant r$ ……**(4)**

Ⅳ． 如果 B 的总空间数 $r(i-1)$多于 A 的总空间数 $x-r$，即 $r(i-1)>x-r$，得 $x<ir$。

又因为 $i \leqslant n-1$，由**(1)**有 $i(r-1)<n(r-1)<x<ir$ 。

因此，原(n,r)-密集形式中至多能填入 r-1 个 i-反馈形式（r=1 是平凡的，这里只考虑 $r \geqslant 2$ 的情况）。即只须证明原(n,r)-密集形式中能填入 r-1 个不同的 i-反馈形式。

直接考虑被原(n,r)-密集形式包含的$(n,r-1)$-完美密集形式。

下面证明$(n,r-1)$-完美密集形式中能填入 r-1 个不同的 i-反馈形式：

想象一个类似 2.2 节中的数表，我们在它的每“列”中填入一个 i-反馈形式，因为它共有 r-1 列，故共填入了 r-1 个 i-反馈形式。这些 i-反馈形式可以无约束地占据一列 n 个空间中的 i 个空间，所以，只要证明 i-反馈形式的总数多于 r-1 个即可。

由**(2)**，i-反馈形式总数 $C_n^i = C_{n-1}^i + C_{n-1}^{i-1} > C_{n-1}^{i-1} - 1 \geqslant r-1$。得证。

Ⅴ． 如果B的总空间数$r(i-1)$不多于A的总空间数$x-r$，结合**(4)**知，A,B满足**引理 3** 的条件。于是由**引理 3**，A可被分成B和另一个（n-1）-密集形式C。由**(2)**，因为B的总空间数$r(i-1) \leqslant (i-1)C_{n-1}^{i-1}$，且 $1 \leqslant i \leqslant n-1$，根据归纳假设，这个密集形式能被$(i-1)$-充分填充，并且用$r$个$i$-反馈形式填充后恰无空间剩余（总空间数是$(i-1)r$）。把$B$被$(i-1)$-充分填充的方案中所有$(i-1)$-反馈形式附加上$X_1$分量后，得到$r$个$i$-反馈形式，它们恰好填满了$B$和$X_1$。这些$i$-反馈形式均有$X_1$的分量，不会与填充$C$的$i$-反馈形式相同。

因此，下面只需证明 C 能被 i-充分填充。

Ⅵ. 考虑 V 中剩下的(n-1)-密集形式 C。

因为 1≤i≤n-1，且剩下的空间数为 $x-r-r(i-1)=x-ir$，

又由**引理 2** 和**(3)**，有

$$x-ir \leqslant x-i\frac{x}{n}=(1-\frac{i}{n})x \leqslant (1-\frac{i}{n})iC_n^i=iC_n^i-i\times\frac{i}{n}C_n^i=iC_n^i-i\times C_{n-1}^{i-1}=iC_{n-1}^i。$$

因此，剩下的鼠群满足归纳假设的条件，可以被 i-充分填充，得证。

综上，$n=k+1$ 时，命题得证。

综合 1)和 2)，原命题得证。

根据**定理 2**，可知 $n, (r-\sum_{i=1}^{Z(n,r)-1}C_{n-1}^{i-1})$-完美密集形式能被 $Z(n,r)$-充分填充，故(n,r)-完美密集形式能被 0-充分填充，即数表-填充方法的可行性获证。

应当注意到，**定理 2** 的证明过程给出了数表-填充方法例子的递归构造法，其核心是先填充一整行并使其他行的空间分布保持均匀，再递归地按少一行的方法填充剩下的部分。

2.4 对实验轮数的改进

在前文中，我们只研究了 n,r 两个变量。而事实上，这个问题还可以加入实验轮数 m，从而进行更深入的推广。

最直接的推广是：用 n 只老鼠进行 m 轮实验，其中每只老鼠每轮实验至多服用 r 瓶药，最多能确保从多少瓶药中选出一瓶毒药。类似 2.3 节中的定义，每轮实验都相当于一个 $n\times r$ 数表，因此 m 轮实验的空间总体相当于一个 $m\times n\times r$ 的三维数表。这时反馈形式也被提升到了三维，并且由于老鼠死后不能再做实验，所以同一只老鼠在两轮不同的实验中所服的药物的集合应当没有交集（否则在前一轮中就已经能判断该药是不是有毒了，这样也起不到作用，参见 1.2.3 节关于实验“独立性”的描述）。因此，对反馈形式的限制是，每个反馈形式只能占据每只老鼠的至多一个空间，即在每只老鼠决定的“平面”内有至多一个分量。

与 1.4 节类似，还可以把问题中“生死状态”换成“叫声状态”。这样做的影响就是对反馈形式的限制变成了每个反馈形式能占据每只老鼠每轮实验至多一个空间，即在每个“老鼠-实验直线”（即老鼠决定的平面与实验轮数决定的平面的交线）上有至多一个分量。

以上两个问题就是 $b(n,r)$问题推广到三维空间后的表现形式。对反馈形式限定的不同决定了它们的差异。同样的，不难把数表-填充方法的各种定义推广到三维空间中，并可以用该方法得到这两个问题答案的上界。如果不用老鼠与毒药的实际化条件而直接采用抽象的数表方法，本问题也可以推广到一般的多维形式。然而，上文中的可行性证明依赖于对(n,r)-密集形式的一种划分，而这种划分无法简单地推广到三维及更高维中。因此，不能确定此时的数表-填充方法上界是否真的能取到。这个问题有待于进一步的研究。

第三章 两瓶毒药问题的推广

3.1 问题的描述

上文中所有方案限于检测仅含有一瓶毒药的问题，那么如何处理多瓶毒药的问题呢?

如果药物中含有一瓶以上的毒药时，喝任意一瓶毒药均会导致老鼠死亡，使问题变得更复杂。这里我们只考虑含有两瓶毒药的情形，并将新问题描述如下：

问题 3 对于充分多瓶药物，已知其中共有两瓶毒药。用 n 只老鼠做一轮实验，最多能确保从这些药物中检验出占多少比例的药物无毒？

下文中，设**问题 3** 中的最大比例为 $c(n)$。

3.2 坐标方案

由于坐标思想的具体方案与 n 的整除性有关，为了在不失一般性的同时更具体的描述这个方案，下面用特例 $c(10)$来介绍它。

3.2.1 基础方案：5×5 方案

将 10 只老鼠平均分成两组，并做出一个 5×5 的方阵，把全部药物均匀地放在这个方阵中。（因为药物足够多，所以可以视作是连续的，而无需考虑无法被 25 整除所带来的微小误差，下同）然后让第一组的 5 只老鼠分别喝方阵 5 横排中所有的药，第二组则喝竖列的。因为毒药所在方格横排和竖列所对应的老鼠都会死亡，从而根据老鼠死亡情况就能判断哪格有可能是毒药。最坏情况是两组各死两只老鼠，如下图所示。此时，这 4 条“直线”将有 4 个交点，都是可能有毒药的方格。也就是说，虽然只有两瓶毒药存在，这里将找出 4 个可能的毒药所在地，发生了混淆。

	6#老鼠（生)	7#老鼠（死)	8#老鼠（生)	9#老鼠（死)	10#老鼠(生)
1#老鼠（生)	无毒	无毒	无毒	无毒	无毒
2#老鼠（死)	无毒	可能有毒	无毒	可能有毒	无毒
3#老鼠（生)	无毒	无毒	无毒	无毒	无毒
4#老鼠（死)	无毒	可能有毒	无毒	可能有毒	无毒
5#老鼠（生)	无毒	无毒	无毒	无毒	无毒

这时，能确保无毒的方格有 5×5－4=21(个)， 确保药物无毒的比例为 $\frac{5\times5-4}{5\times5}=\frac{21}{25}$，

即 $c(10)\geqslant\frac{21}{25}$。

本节的平均分组是为了使得方阵的方格数最多(5×5 比 4×6 或其他分组方式乘积更大)，从而得出的答案最优。

3.2.2 改进方案Ⅰ：6×6 方案

正如进位制方法中可以有编号为 00…0 的药物一样，这里也可以考虑使用排除的思想。

把药物分成 6×6 的方阵，并只让两组老鼠分别喝前 5 横排和前 5 竖列的药物。如果某组只有一只老鼠死亡，我们就认为那只老鼠所确定的排（列）和第 6 排（列）都可能存在毒药。如下图所示。与 3.2.1 节相同，最坏情况下是有 4 格里可能有毒药。

	6#老鼠 (生)	7#老鼠 (死)	8#老鼠 (生)	9#老鼠 (死)	10#老鼠(生)	
1#老鼠 (生)	无毒	无毒	无毒	无毒	无毒	无毒
2#老鼠 (死)	无毒	可能有毒	无毒	可能有毒	无毒	无毒
3#老鼠 (生)	无毒	无毒	无毒	无毒	无毒	无毒
4#老鼠 (生)	无毒	无毒	无毒	无毒	无毒	无毒
5#老鼠 (生)	无毒	无毒	无毒	无毒	无毒	无毒
	无毒	可能有毒	无毒	可能有毒	无毒	无毒

综上，确保无毒的比例是 $\frac{6\times6-4}{6\times6}=\frac{8}{9}$，即 $c(10)\geqslant\frac{8}{9}$。

3.2.3 改进方案Ⅱ：4×4×5 方案

事实上，上述方案的主要思想是设定 2 个独立的坐标，就像平面直角坐标系中用两个数表示一个点的位置一样。因此，容易联想到做出类似空间直角坐标系的三维方案，以及更一般的任意多个坐标的方案，这就是坐标方案的核心思想。

作为三维方案，把 10 只老鼠尽量均匀的分为 3 组，即 3 只、3 只、4 只。考虑到 3.2.2 节中的改进，可以把药物分成 4×4×5 的空间阵列并按上述方法实验。每组老鼠代表着一个坐标。平面直角坐标系中用线性方程表示一条直线，所以老鼠喝一条直线上的药；而空间直角坐标系中线性方程表示一个平面，所以老鼠需要喝一个平面上全部的药物。这时，又有一个更复杂的混淆问题，因为最坏情况是将会出现每组两只老鼠死亡，就相当于确定了毒药所在的 3 组平面，其中每组内两平面平行。这 6 个平面有 8 个交点，即有 8 格无法确定毒性。（一般的，直接考虑n个坐标、每个确定 2 个超平面可知共有 2^n格可能有毒药）因此，确保无毒的比例为 $\frac{4\times4\times5-8}{4\times4\times5}=\frac{9}{10}$，即$c(10)\geqslant0.9$。

因为无法确定毒性的方格数量的指数级增加，添加出更高维或减少至一维均不能使无毒比例增加，所以，这个方法所得的 n=10 时的最好下界就是 0.9。对于 n 取一般数值的问题，类似的可以依次考虑二维、三维及更高维的坐标方案，并取其中的最优解。

3.3 坐标-分组方案

还有一种看起来较简单的方案，就是直接把药物平均分成几组，并把每组考虑成一瓶药物来减少独立药物的数量，然后让每只老鼠喝下其中的若干组，就像二进制方案一样，通过

死亡情况来找出包含药物的两组（因为只考虑最坏情况，假如两瓶毒药不幸被分到一组，允许只找出包含毒药的两组，而不确定是否毒药在哪组。这种特例有可能对下面的研究产生影响，因此要特殊注意。这个特殊情况以下称为蕴含问题）。

经过初步分析容易发现，要确定分组方法的答案是困难的。

为此，以下构造一个新问题：

问题 4 在已知有两瓶毒药的情况下，用 n 只老鼠进行一轮实验，最多能确保从多少瓶药物中找出这两瓶毒药？

我们设**问题 4** 的解为一个新函数 $f(n)$。

把上面问题中的药物理解成组，可知分组方案所得到的确保无毒比例为 $\frac{f(n)-2}{f(n)}$。

事实上，到目前为止，我们还没有证明存在 n，使得 $f(n)>n+1$（n 只老鼠各检验 1 瓶（组），最后一瓶（组）用排除法，就得到 $f(n)\geqslant n+1$），而且简单的尝试可以发现，构造一个非平凡的例子是非常困难的。同时，要想使得这个方法所得到的界优于坐标方案，需要 $\frac{f(10)-2}{f(10)}>0.9$，即 $f(10)>20$，所以，从这个意义上看，这个方法势必不如坐标方案，要想有更优的结果，需要更深刻的想法。不过实际上，这个分组方案是可以"整合"到坐标方案中的！这是因为坐标方案中 5 只老鼠被用来从 6 组药中选出（可能）有毒的两组，而分组方案中这个数应该是 $f(5)\geqslant 6$。因此，二维情况可以是 $\frac{f(5)^2-4}{f(5)^2}$，同理三维及更高维也有改进空间，可能还会优于坐标方案给出的答案。因此，在坐标方案中每组的 x 只老鼠不止通过各喝一个超平面上的药物、而是按分组方案设计，则可能会用来检验更多超平面，于是我们就将这两个方案合二为一，称为坐标-分组方案。

另外，这里还存在一个问题，就是 3.2.1 节中，我们尽量使每组老鼠数量相同是为了让它们的乘积最大，而对于函数 f 来说，这样是否同样能达到最大值呢？三维又是否是最优的呢？为了解决上面的诸多疑问，我们必须对 $f(n)$进行更深入的研究。

3.4 对 f(n)的定性估计

3.4.1 二进制描述

为了方便进行下面的研究，我们将寻找一种合适的描述方法。

设计例子的关键就是哪只老鼠喝哪些药，为了描述这个情况，我们采取类似第一章中的二进制方法：每行表示一瓶药，每列表示一只老鼠，第 j 只老鼠喝第 i 瓶药则第 i 行第 j 列为 1，否则为 0。比如说，3.2.1 节中第一组的老鼠服药情况如下：

$$\begin{matrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{matrix}$$

我们把这个方阵按横排分成 n 个部分（即 n 瓶药），用这样的表示法，将一个服药方案转化成了 $f(n)$个 n 位二进制数。联系第二章关于反馈形式的定义，我们也称这样的二进制数为对应药物的反馈形式。

3.4.2 "合成"运算和识别条件

在第二章中，为了在药物中找出哪一瓶有毒，只需让每瓶药物的反馈形式不同；而现在需要找出两瓶毒药，就得采用更复杂的方式来判断给定的反馈形式是否满足题目要求（即是否满足所谓的"识别条件"）。为了确切的找出两瓶毒药，当任两瓶药是毒药时，老鼠的死亡情况都应该不同。为了对老鼠死亡情况的描述，我们引入"合成"运算：

定义 3.1 or（合成）运算：对于两个位数同为 n 的二进制数 a,b，它们的合成也是位数为 n 的二进制数，其中每个数位是原来两个数对应数位上数字中较大的那个，这个新数记作 ***a or b***。

这个定义确保了当且仅当两个数中至少一个的某位为 1，则它们合成结果的相同数位为 1。这个运算其实就是计算机位运算中的"或"。理解为老鼠与毒药的位置关系，就是当这两瓶药为毒药时，当且仅当其中至少一瓶药被某只老鼠喝，这只老鼠的反馈为 1，即死亡。这样，我们就得到了对两瓶毒药时情况的描述。由此，为了能区分任两瓶药是毒药的情况，就要求任两个二进制数的合成不同，这就是新问题中的识别条件。所以，我们的问题化为求至多能找出多少个 n 位二进制数，使其两两的合成均不同。

3.4.3 f(n)上界

$f(n)$个n位二进制数两两做合成运算，共得到$C_{f(n)}^2$个n位二进制数。为使它们两两不同，这个数量一定少于n位二进制数的总数 2^n。因此，$C_{f(n)}^2 \leqslant 2^n$。

解得 $f(n) \leqslant \dfrac{1+\sqrt{1+2^{n+3}}}{2} < \dfrac{1+(1+\sqrt{2^{n+3}})}{2} = 1+\dfrac{\sqrt{2^{n+3}}}{2} = 1+2^{\frac{n+1}{2}}$

综上，可以得到 $f(n)$的指数级上界。

3.4.4 单调性

假设我们已经得到了满足条件的 $f(n)$个 n 位二进制数，把这些数加上末位 0 得到 $f(n)$个新数。再加入 n+1 位二进制数 00…01（前 n 位为 0），这 $f(n)$+1 个新数将满足任两个的合成互不相同。这是因为末位是 0 还是 1 决定了参与合成的两个数中是否包含 00…01，而无论包含与否，我们的"归纳假设"（即原来的 $f(n)$个数满足识别条件）保证了所有的合成两两不同。

综上，得到了 $f(n)$+1 个满足条件的 n+1 位二进制数，故 $f(n+1) \geqslant f(n)+1$，即 $f(n)$是严格单调递增的。

3.4.5 f(n)下界与一个重要的函数不等式

从数量级估计的角度来讲，上节的不等式是平凡的。不过它提示了我们，用 n 是某个数值时的例子去构造 n 更大时的例子，从而得到下界。

定理 3 $f(n+2\lceil\log_2(n+1)\rceil)\ge 2f(n)$（其中$\lceil x\rceil$表示不小于 x 的最小整数，下同）

证明：假设我们已经得到 $f(n)$个 n 位二进制数，它们两两的合成互不相同。

令 $k=\lceil\log_2(n+1)\rceil$，下面，构造 $2f(n)$个满足识别条件的 $n+2k$ 位二进制数。

依次构造每组 $f(n)$个的两组 $n+2k$ 位二进制数。将原来的 $f(n)$个 n 位二进制数后面加上 k 个 0，再分别加上 k 个数位，表示 1,2,…,n 的二进制编号（因为 n 的二进制表达恰好需要 k 个数位，所以这是可以办到的）来构造第一组。第二组数只需将第一组中后 $2k$ 位中前 k 位和后 k 位调换过来即可。

这 $2f(n)$个数两两的合成中，根据后 $2k$ 位的情况，可知两个参与合成的数取自上面构造的哪组：如果后 $2k$ 位中，前 k 位为 0，则两个数都来自第一组；后 k 位为 0 则都来自第二组；两种都不是则一个来自第一组，一个来自第二组。如果两个数来自相同的组，“归纳假设”决定了这些数满足识别条件，否则，把后 $2k$ 位直接看做两个 k 位二进制数，并转化为十进制数就可以得到两个数在各自组中的编号，即可唯一确定地找出这两个数，从而就证明了这时的结果也满足识别条件。

综上，我们构造出 $2f(n)$个满足识别条件的 $n+2k$ 位数，也就证明了**定理 3**。

3.4.6 f(n)定义域的连续化和进一步的下界估计

上节中使用了离散化的天花板函数 $\lceil x\rceil$，并且 $f(n)$也是定义域离散的函数。这些离散属性在函数的处理中很不方便。为此，本节将试图在不影响其效果的前提下把它们连续化。

对任意 $x\in[1,+\infty)$，为将函数 f 的定义域连续化，定义 $f(x)$为原来的 $f([x])$，则新的函数仍是单调递增的（但不是严格单调递增的），且在$[1,+\infty)$上有定义。

$\because [2\log_2(n+1)]+2=[2(\log_2(n+1)+1)]\geqslant[2\lceil\log_2(n+1)\rceil]=2\lceil\log_2(n+1)\rceil$

$\therefore 2f(n)\leqslant f(n+2k)\leqslant f(n+[2\log_2(n+1)]+2)=f(n+2\log_2(n+1)+2)(n\in\mathbf{N}_+)$

同样的，对于正实数n，我们有 $f(n+2\log_2(n+1)+2)\geqslant f([n]+2\log_2[n+1]+2)\geqslant 2f([n])=2f(n)$,

以上计算得出了对**定理 3** 的一个改进，从而不但去除了取整函数，还使得n可以取任意实数值。下节我们将运用 $f(n+2\log_2(n+1)+2)\geqslant 2f(n)(n\in[1,+\infty))$ 来估计$f(n)$的下界。

3.4.7 f(n)下界与幂函数的比较

设 $h(n)=\dfrac{n+2\log_2(\mathrm{n}+1)+2}{n}$，则有

$$h'(n)=\frac{2}{n}[\frac{\log_2 e}{n+1}-\frac{1}{n}-\frac{\log_2(n+1)}{n}]<\frac{2}{n^2}(\log_2 e-1-\log_2(n+1))\le\frac{2}{n^2}(\log_2 e-1-1)<0\ (n\in[1,+\infty)),$$

所以，$h(n)$单调递减。

因此，对于$n>n_0\geqslant 2$，有　$f(h(n_0)n)\geqslant f(h(n)n)=f(n+2\log_2 n+2)\geqslant 2f(n)$.

故　$f(h^x(n_0)n_0)\geqslant 2f(h^{x-1}(n_0)n_0)\geqslant\cdots\geqslant 2^x f(n_0)(x\in\mathbf{N}_+)$。

设 $y=h^x(n_0)n_0(x\in\mathbf{N}_+)$，则有　$x=\log_{h(n_0)}\frac{y}{n_0}$，代入得

$f(y)\geqslant 2^x f(n_0)=f(n_0)\,n_0^{\log_2 h(n_0)}y^{\log_{h(n_0)}2}=C\,y^{\log_{h(n_0)}2}$　　($C=f(n_0)n_0^{\log_2 h(n_0)}$，是与$n_0$有关的常数)

推广到充分大的任意实数 y，因为上面整数的限制，有

$$f(y)\geqslant 2^{[\log_{h(n_0)}\frac{y}{n_0}]}f(n_0)>2^{\log_{h(n_0)}\frac{y}{n_0}-1}f(n_0)=\frac{C}{2}y^{\log_{h(n_0)}2}。$$

因为　$\lim\limits_{n\to\infty}h(n_0)=\lim\limits_{n\to\infty}\frac{n+2\log_2(n+1)+2}{n}=\lim\limits_{n\to\infty}\frac{(n+2\log_2(n+1)+2)'}{(n)'}=\lim\limits_{n\to\infty}(1+\frac{2}{n+1})=1$（洛必达法则），所以，$\log_{h(n_0)}2$ 可以任意大。故 y 足够大时，$f(y)$增速可以超过任何幂函数。

因为上节的不等式中既有 n 又有对数，所以难以给出确切的数量级。假如去掉对数，得到类似 $f(n+2)\geqslant 2f(n)$ 的式子的话，则容易算出函数恰是指数级的。

所以，我们可得出 $f(n)$下界介于指数级和幂函数级别之间。

3.5　对 f(n)的定量计算 (计算机算法)

3.5.1 精确解的搜索算法

为了求出问题的精确解，我们设计了依靠回溯算法进行求解的程序。

从空方案开始，每次按二进制数从小到大枚举每个反馈形式，尝试添加进待求方案中。若新反馈形式添加后满足识别条件，则继续枚举添加下一个。无法添加任何一个新反馈形式时，删除最后一个添加的反馈形式。如此直至枚举完所有合法的方案，从中取反馈形式最多的一个进行输出。这里，为了提高程序效率并方便处理，直接将反馈形式存储为对应的二进制数，并用位运算“或”处理合成运算。（Pascal 源程序见附录 A）

这个程序能保证解得$f(n)$的最优性。不过单从枚举反馈形式的角度讲，它的时间复杂度就高达$o(2^n)$。因此，该程序所用的时间随着n的增加而急剧增多，不适合n较大时的求解。

在实践中，我们通过这个程序得到了 $n\leqslant 8$ 时的解，见下表（对应方案见附录 C）。其中计算 $n=7$ 的过程不到一分钟时间，而 $n=8$ 时，则用了几个小时，可见其时间消耗增速之快。

n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
***f*(*n*)**	**2**	**3**	**4**	**5**	**6**	**8**	**10**	**13**

3.5.2 快速估计的随机数算法

由于上面算法的时间复杂度难以承受，我们决定尝试新的策略，使用随机方法在相对较短的时间内求出尽量接近 $f(n)$的估计解。

从空方案开始，每次从可以添加的反馈形式中随机选取一个，添加进待求的方案，同时记录本次求解过程中曾得到的最优解。当不能再添加新数时，则从当前方案中随机删去若干个反馈形式并重新尝试添加。如果得到了更好的结果，就输出并继续上述过程；如果进行了一定次数的操作仍不能改进答案，则放弃当前解，再从空方案开始。（Pascal 源程序见附录 B）

这个程序的运行效果比较好，能很快的算出 n 在一定规模时的解，$f(n)$估计值见下表（对应方案见附录 C）。不过，由于所得解不一定是最优解，这仍然不是理想的解决途径。

n	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
***f*(*n*)估计值**	**17**	**21**	**30**	**37**	**42**	**52**	**64**	**81**	**102**	**126**	**159**	**195**

3.6 对 f(n)的猜想与展望

经过以上的研究，我们对 $f(n)$在 n 比较小时的数值及比较大时的变化趋势都有了更深入的了解。然而许多问题仍未得到解决。

最为关键的一个问题就是 $f(n)$的数量级。在 3.4 节中，我们得到的上界是指数级的，而下界则达不到这个等级。鉴于 n 比较小时的数据与下界估计已有差距，我们猜测 $f(n)$是指数级的。不过要想从估计下界的角度得到这一点，似乎需要更为深刻的构造方法以便得到更强的不等式。

$f(n)$的“平稳”性是另一个值得关注的问题。注意到由 3.5.1 节，对 n=1,2,…,7，$g(n)=f(n+1)-f(n)$是单调递增的。单从定义来看，也没有理由认为存在某个特定的 n 使得 $f(n)$发生突变，因此我们猜测 $g(n)$恒单调递增。不过，3.5.2 节得到的结果则似乎否定了这一点，假设 3.5.2 节算出的是精确解，则有 $g(12)<g(11)$。这究竟是因为刚刚的猜测错了还是 3.5.2 节的程序得到的答案不具备最优性所致，我们还不清楚。

例子的构造同样是个有趣的问题。3.5.1 节中我们由程序得到的例子全都含有一个共同的反馈形式“00…0”，这是否是必然的呢（3.5.2 节中的例子有不含这个反馈形式的，但不一定是最优解）？并且 3.5 节的程序只给出找到的第一个最优方案，那么对于 n，又有多少组满足条件的最优方案呢？

合成运算在实现时用的是计算机位运算*or*，那么如果“规则”不再是*or*，而是其他位运算呢？根据*and*运算与*or*的对称性知，只要把得到的反馈形式的 0 和 1 互换，它们就是等价的。不过，对于在编码理论中使用最多的*xor*，竟会得到不同的答案。例如把 3.5.1 节程序中*or*改成*xor*，得到新的结果$f^*(6)=9>8=f(6)$。为什么会这样？*xor*和*or*时的结果是否有联系呢？

3.7 蕴含问题

回到坐标-分组方案，考虑到每组包含多瓶药物时，可能有两瓶毒药同在一组的情况，我们的检验法要有一点修改：假如没有两个反馈形式的合成和真实反馈相同，那必定是由于毒

药只在一组中，从而直接由真实反馈形式找出那组即可。

3.3 节中我们论证了只要能找到包含这组的两组即可，然而这时的两组是否唯一呢？是否会存在三组所对应的反馈形式 a,b,c 使得 a or $b=c$，从而，如果毒药都在 c 中会与分别在 a 和 b 中的混淆呢？

我们就此证明：上面的检验法能唯一确定地找出包含毒药的至多两组。

假定一种方案使得 $f(n)$瓶药的反馈形式满足识别条件，即任两个的合成不同。于是，如果毒药分别在两组中，可以唯一确定的找出与真实反馈相同的两个反馈形式的合成，也就找出了相应的两组。如果毒药都在一组，假如没有两个反馈形式的合成和真实反馈相同，我们能判断出这一点，从而直接由真实反馈形式找出那组；否则，有两个反馈形式的合成与真实相同时，会发生所谓的混淆，这时只需要证明找出的两组包含了有毒药的一组，而不是 a or $b=c$ 型的混淆情况。这可以用反证法证明。因为假设选出组的反馈形式为 a,b，真实组的为 c，且 a or $b=c$，其中 c 与 a,b 互不相同，那么 a 中所有为 1 的数位在 c 中所对应数位也为 1，因此，a or $c=c$,，而 $c=a$ or b，因此，a or $c=a$ or b，即 a,b,c 不满足我们构造中任两个反馈形式的合成不同的条件，矛盾！因此 c 和 a，b 之一相同，即找出的这两组包括了含毒药的组。证毕。

事实上，这种方法也可以推广到 n 瓶毒药的情况，即 n 瓶毒药的情况中，假如全部毒药只在少于 n 组中，按上面的检验方法(先找 n 组的合成，没有对应的话再找 n-1 组的，以此类推)处理就可以唯一确定的找出至多 n 组包含全部有毒组。这里要注意还有一种混淆的方式：毒药在 i 组中，但我们寻找 $j(i\leqslant j<n)$个反馈形式的合成时得到不止一种方式。当然这种混淆同样是不可能的，因为从这两种方式中任取一个只在一个方式里出现的反馈形式，取它和另一个方式中所有反馈形式的合成也满足识别条件，因此在 j+1 个的方案中就该出现合适的选法，得到矛盾。

3.8 对坐标-分组方案的新理解

根据 3.5 节的计算，我们考虑 n=10 时的两种二维坐标-分组方案：

第一种是(5,5)分组，最多把所有药分成 $f(5)\times f(5)=36$ 组，能鉴定无毒药物的比例是 $\frac{6\times6-4}{6\times6}=\frac{8}{9}$。第二种是(4,6)分组，把药分成 $f(4)\times f(6)=40$ 组，能鉴定的比例是 $\frac{40-4}{40}=\frac{9}{10}>\frac{8}{9}$。因此，对二维坐标而言，(4,6)分组优于(5,5)分组，故 3.2.1 节中通过平均分组来最优化的想法不再成立。

进一步尝试可知，(3,7)、(2,8) 分组也都优于(5,5) 分组，并且未呈现什么好的数据模式。因此，似乎很难判断对特定的 n，什么样的分组更合适。另外，对三维及更高维进行尝试得到最优比例仍仅是 0.9。所以，对 n=10 的特定情况，高于一维的坐标-分组方法仅限于给出确定 90%的药物无毒。可是，对于一维的纯粹分组方案，由 3.5.2 节的数据，有 $f(10)\geqslant21$，由此得到 $c(10)\geqslant\frac{21-2}{21}=\frac{19}{21}>\frac{9}{10}$！因此，坐标方案是否有意义也成了未知数。

当n充分大时，若仅用指数级别的连续函数近似处理 $f(n)$，比如设 $f(n)=ab^n$，P维时分组方案为$m_1,m_2,\dots,m_p$，其中$\sum_{i=1}^{p}m_i=n$，则把药物分成 $f(m_1)f(m_2)\cdots f(m_p)=a^p b^{\sum_{i=1}^{p}m_i}=a^pb^n$份，只与维数有关！因此，确保无毒的比例为 1-$\frac{2^p}{b^n a^p}$=1-$b^{-n}(\frac{2}{a})^p$。如果$a>2$，则维数越多越好；$a<2$ 时，

一维最好；a=2 时，多少维无所谓。那么，如果$f(n)$真是指数级的，至少维数固定时分组的方式就不重要了，本节n=10 时的讨论也只是一个平凡的特殊数值问题了。不过，如果$f(n)$是下界估计中的亚指数级别，类似的计算将变得非常复杂，整个坐标-分组方案的效果也将很难预测。

3.9 c(n)上界估计与信息论方法

在**问题 3** 中，我们采用“充分多瓶”药，避免了整除性带来的繁琐讨论。不过，这也使得 $c(n)$的上界估计成了一个困难的问题。为此，我们必须找到应对“充分多”的统计方法来研究 $c(n)$。信息论中信息熵的概念恰好提供了这种方法。

对于未知的不确定事物，我们用概率来进行描述。例如，x 瓶看似相同的药物中有两瓶毒药，我们就视作药物有毒的概率为$\frac{2}{x}$。故根据信息论的观点，假设药物可以看做独立的二值信源，则每瓶药物的信息熵为 $E_1 = -p\log_2 p-(1-p)\log_2(1-p)$ 比特，其中 $p=\frac{2}{x}$为该药是毒药的概率。因此，所有药物总的信息熵为 $E = xE_1 = 2\log_2\frac{x}{2}+(x-2)\log_2\frac{x}{x-2}$ 比特。

经过某些服药方案的判断后，有 $xc(n)$ 瓶药物被确定无毒。假定对剩下的药物一无所知，则剩下的药物中毒药分布可视为随机。用上段的方法，可得检验后的信息熵为 $E' = 2\log_2\frac{[1-c(n)]x}{2}+\{[1-c(n)]x-2\}\log_2\frac{[1-c(n)]x}{[1-c(n)]x-2}$ 比特。

因为 x 充分大，可以把$\frac{x}{x-2}$和$\frac{[1-c(n)]x}{[1-c(n)]x-2}$近似地当作 1。

两个信息熵的差值$E-E'$约为$2\log_2\frac{x}{2}-2\log_2\frac{[1-c(n)]x}{2}=2\log_2[1-c(n)]$ 比特。这个差值就是我们获取的信息的度量。

另一方面，我们获取的信息来自 n 只老鼠。由于方案的任意性和毒药的随机性，这里视它们生死概率相同（类似 1.3 节）。因此每只老鼠恰好是理想的二值信源，各提供 1 比特信息。所以，老鼠提供的信息总量为 n 比特。

因此，有 $2\log_2[1-c(n)]\le n$，解得$c(n)\le 1-2^{-\frac{n}{2}}$。

考虑纯粹的分组方法，假设它能达到这个上界，即$1-2^{-\frac{n}{2}}=\frac{f(n)-2}{f(n)}$，解得 $f(n)=2^{\frac{n+2}{2}}$，它很接近 3.4.3 节中的 $f(n)$上界。

由于上面的论证中有大量独立化、理想化的假设，并忽略了信息熵的一部分差值，所以，这个 $c(n)$上界并不是准确的。同时，我们对坐标-分组方案的鉴定能力也缺乏精确描述。因此，坐标-分组法究竟是不是这个问题的最优解还有待于进一步研究。

总结与展望

通过以上三章的研究，我们得到了对经典老鼠与毒药问题的三个推广，其核心内容分别由三个定理体现：

首先，把原始问题对时间进行推广后，在利用进位制解法的探索中，隐含的离散化条件和时间与实验轮数的本质关联就显现了出来。**定理 1** 及其证明则标志着这个问题的完美解决，也使我们对整个问题有了更深入、更彻底的认识。

其次，在原始问题中加入一个限定条件后，我们得到了一个新的推广并设计出了数表-填充方法予以解决。看似明显实则困难的**定理 2** 既是上述问题的结果，也是新猜想的开始。在这个问题中加入实验轮数后，我们得到了三维数表的两个版本，从而开启了一个更高维问题的新领域。

最后，我们研究了最为困难的两瓶药物问题。从坐标方案到分组方案再到坐标-分组方案，我们提出并研究了 $f(n)$。对这个函数的定性、定量分析充满了困难和不确定，这极大的妨碍了完整彻底掌握它的企图。不过，通过估计和计算机编程，我们最终得到了定理 3 和一定规模下的具体数值。这些工作开拓了该问题的主要思路与方法，并在给出深刻结果的同时提出了一系列新的思考。

虽然这三个推广都是以组合趣题的形式出现，但是在其他领域也有其现实意义。前两个推广可以用来优化特殊类别的生物实验（一般的生物实验为了避免测试“药物”的互相干扰，采用严格的控制变量法；而在太空等极端环境中，效率则相对更为关键）或用于检验商品、货物质量等。最后一个推广则更倾向于纠错码的研究，将问题中的“规则”or 改成 xor，该问题就变成了一个一般性编码问题（“检验两瓶药”也就相当于“能纠两位错误”），从而类似的思考不难推广到编码理论中。

回顾整篇论文，如此简单的一个组合趣题竟能激发这么多的思考，也着实让我们吃惊。事实上，还有千千万万这样的简单问题背后隐藏着巨大的发展空间，等待着我们去发现、去研究。通过这次美妙的数学旅程，我们收获了知识与经验，也体会了数学的魅力和研究的乐趣。

致谢

首先，感谢东北育才学校的张雷老师和彭玲老师，他们对课题研究予以指导和支持。其次，感谢郑连伟老师，帮助我们按照本科生学位论文的格式进行规范书写和排版，使得我们的论文更加清晰明了。最后感谢家长，他们的鼓励与支持给予了我们极大的勇气和力量。

由于知识有限，底蕴不足，论述中一定有许多不恰当之处。我们本着锻炼、学习的目的，认真完成了论文，希望老师和同学们不吝赐教，谢谢！

附录 A 3.5.1 节源程序

```
var
  f:array[0..1048575]of boolean;                    {f:记录哪些“并”已经存在}
  d,d2:array[1..10000]of longint;   {记录 01 串集合，d 为过程中使用的变量，d2 为最终解}
  n,ans,i,j:longint;
procedure gen(k,lim:longint);
  var                                   {k:当前搜索集合中的第 k 个串  lim:从哪个 01 串开始}
    i,j,t:longint;
    f1,f2:boolean;
  begin
    if k-1>ans then begin                                          {找到更优解，记录}
      ans:=k-1;
      writeln(ans);
      d2:=d;
    end;
    for i:=lim to (1 shl n)-1 do begin                               {枚举第 k 个串}
      f1:=true;
      for j:=1 to k-1 do                                             {判断是否冲突}
        if f[i or d[j]] then begin
          f1:=false;
          break;
        end;
      if f1 then begin
        f2:=true;
        for j:=1 to k-1 do begin
          if f[i or d[j]] then begin
            f2:=false;
            t:=j;
            break;
          end;
          f[i or d[j]]:=true;
        end;
        if f2 then begin                    {无冲突，将当前串添加至集合并尝试添加下一个串}
          d[k]:=i;
          gen(k+1,i+1);                                                 {递归调用}
          for j:=1 to k-1 do
            f[i or d[j]]:=false;
        end
        else for j:=1 to t-1 do
          f[i or d[j]]:=false;
      end;
    end;
  end;
begin                                                                    {主程序}
  assign(input,'in.txt');
  reset(input);
  readln(n);
  close(input);
  ans:=0;
  fillchar(d,sizeof(d),0);
  fillchar(d2,sizeof(d2),0);
  fillchar(f,sizeof(f),0);
  gen(1,0);
  assign(output,'out.txt');
  rewrite(output);
  writeln(ans);
  for i:=1 to ans do begin                                                {输出}
    for j:=n-1 downto 0 do
      if odd(d2[i] shr j) then write(1)
      else write(0);
    writeln;
  end;
  close(output);
end.
```

注：在文件 in.txt 内输入待求的 n，最终答案在 out.txt，计算过程中显示当前得到的最好数值。

附录 B 3.5.2 节源程序

```
var
  f,f2:array[0..20000000]of boolean;
  d:array[1..10000]of longint;
  n,i,j,t,z,h,ans,mt,ct,fc:longint;
  fo:text;
function check(m:longint):boolean;                          {判断新加 01 串是否冲突}
  var
    i,j:longint;
  begin
    for i:=1 to z do begin
      if f[m or d[i]] then begin
        for j:=1 to i-1 do
          f[m or d[j]]:=false;
        exit(false);
      end
      else f[m or d[i]]:=true;
    end;
    for i:=1 to z do
      f[m or d[i]]:=false;
    exit(true);
  end;

procedure add(m:longint);                                   {添加串}
  var
    i:longint;
  begin
    for i:=1 to z do
      f[m or d[i]]:=true;
    inc(z);
    d[z]:=m;
  end;

procedure delete(p:longint);                                {删除串}
  var
    i:longint;
  begin
    t:=d[p];
    d[p]:=d[z];
    d[z]:=t;
    for i:=1 to z-1 do
      f[d[z] or d[i]]:=false;
    dec(z);
  end;

begin
  randomize;
  readln(n);
  ans:=0;
  while true do begin
    fillchar(f,1 shl n,0);
    fillchar(f2,1 shl n,0);
    z:=0;
    mt:=0;
    fc:=0;
    while true do begin
      ct:=0;
      for i:=0 to (1 shl n)-1 do                            {随机选取可添加的串}
        if not f2[i] then if check(i) then begin
          inc(ct);
          if random(ct)=0 then h:=i;
        end
        else f2[i]:=true;
      if ct=0 then begin                                    {无法添加，删除串}
        for j:=1 to 5 do
          if z<>0 then delete(1+random(z));
        fillchar(f2,1 shl n,0);
      end
      else add(h);
      if z>mt then begin                                    {更新本次最优解}
        mt:=z;
        fc:=0;
      end
      else begin
        inc(fc);
        if fc=mt*10 then break；                            {超时，放弃}
      end;
    end;
    if z>ans then begin                                     {更新全局最优解并输出}
      writeln('# ',z);
      ans:=z;
      assign(fo,'result.txt');
      rewrite(fo);
      writeln(fo,n,' ',z);
      for i:=1 to z do begin
        for j:=0 to n-1 do
          if odd(d[i] shr j) then write(fo,1)
          else write(fo,0);
        writeln(fo);
      end;
      close(fo);
    end;
  end;
end.
```

注：这个程序不能自动结束，需要手动终止。打开程序后直接输入待求的 n，结果在 result.txt 中。

附录 C　n≤20 时的 f(n) 对应方案

n≤5 时，f(n)=n+1，是平凡的。

n≤8 时，方案由 3.5.1 节程序给出：

n **=6 8**	000000 000011	001100 010101	011010 100110	101001	110000
n **=7 10**	0000000 0000011	0000101 0001001	0010010 0100100	0111000 1001000	1010100 1100010
n **=8 13**	00000000 00000011 00001100	00010101 00100110 00111000	01001001 01010010 01100000	10001010 10010000	11000100 10100001

9≤n≤20 时，方案由 3.5.2 节程序（不一定是最优解）给出：

n **=9 17**	100000011 011000110 000110011 000101001	101010010 000010101 010010000 000001100	100011000 001100000 011011000	010110100 101000100 011000001	110100000 000000010 010101010
n **=10 21**	0101001001 1010100001 0001010010 0101100000 0110110000	0000000000 1100000100 1000110000 0000011001	0010011010 0100000011 0000100010 0110000110	0001001100 0000010100 1001010001 0000100101	0010101100 1011000000 0110001000 1100001010
n **=11 30**	01000000000 11000010010 01000010101 00010100110 01110000100 01011010000	00000001111 00000110011 01001100010 00110001001 00001101100 11101000000	10110000010 00110110000 10010000101 10000110100 11010100000 00101011000	01010001010 00010011100 00101000101 00001010110 00100101010 01000111000	10000101001 00011100001 10001010001 10001001010 01001001001 01100100001
n **=12 37**	001000101001 100010000110 101001001000 101000000011 001011000010 000110010100 100100000101 100010101000	110001000100 001101000100 010101001000 000011100100 010000011001 010011010000 001001110000 110000100001	000100010011 011100010000 001000011010 100000011100 100000110010 100111000000 000101100001	010000000111 001110000001 101010010000 100001010001 000100001110 000010001101 101100100000	010000101100 011010001000 010001100010 011001000001 100000000000 000010110001 010110100000
n **=13 42**	0000001000010 0110011010000 0010110001000 0000010010011 1010001000100 0001101001100 1100100001000 1000010011010 1000100000011	0000010111000 0001000100010 0000000011100 0010110100001 0101000110000 1000001101000 0000110000010 0000000110111 1001001010001	0011011000001 1110000000010 1000110000100 0110101000000 0100010000001 0100000001110 0001011010100 0011000110100	0000101011010 0000011001101 0001000001011 0100100100101 0011100010010 0101001000011 1101110000000 0111000011000	0100011100010 0000101110000 0000000000101 1100000010101 0010010000110 1001000000000 1010000110001 0010000000000
n **=14 52**	01000010000011 00100000000001 01111000000001 10110000001100 00010101001001 00010000000111 10001100011000 11010100100010 01001000110101 00001001000101 11000101000001	00010010011001 10110001010000 00000100011101 00100001100011 10000000110001 00000000101100 01100100101001 00101000110010 01010101010000 01000000010110 01001011001000	00011000001010 10010001100100 00000001011011 00100101001100 00001101010010 10101001000010 10000110000110 10010010000000 01001010011100 11001010000010	01110001000010 00000110110000 00110100100100 00011000010100 00100100001010 00011011000001 00100010100100 11000000001011 10101110000000 01000111000100	00010010101010 00001110000111 01010000111000 10001001101000 01101001010000 00100010010010 00000000010000 11100000100000 00001100100110 01001101100000

	110101000010001	011011000010001	010000000110100	011100000010110	010011110000010
	100110010010010	110010001000001	001010000000010	001000100001001	001010111000000
n = 15	100010100011000	000001110001000	001100001100001	001000100110011	100000000001111
	011001100010010	000010000001100	100010011000100	000100001010101	000101001000110
64	110001000001011	011100000001100	001111100100000	101001000000000	110101001100000
	000100110110000	010010110010100	001101010000100	001001000101010	100101100000010
	000100000000011	000010101100010	000110100000101	010000011000110	110000100100000
	100111000000100	100001000011100	000111001001001	000000111001101	101000101010001
	101100110000001	010000010100011	000011011000001	100010000110110	000100000101000
	010100001001010	001010010100101	111100010000000	010010010101000	000010001111000
	111000001010010	000001100010101	000000110000110	010110100000000	001000011101000
	000001111100100	001101000011000	110000010011001	100001011110000	001011001010100
	100100101001000	010001001011000	000000000010000	100000100100111	
	1001010101000000	1000010011001000	0010100010010101	1000100011000100	0000000101100011
	0000001001001001	0010100100100000	0100111000000010	0001110001000110	0110010100001100
	0100101010010000	1000010000001011	0000001001111100	0000110111000001	1100001000000011
n =16	1100000010101010	0100010001100100	0011001001000100	0010010010100011	0001000100011001
	0000101010100100	1101110000001000	1000001110000010	0101101000001100	0100110000010100
81	0010000011011100	0001000000110100	0000010100000010	0010001000011010	0000000000001000
	0011101110000000	0100001000100110	0100000110000110	0010110001001000	0010100101010010
	1010000110000001	0000101100010011	1100100001101000	0010001100010100	0000010001111000
	0000010000010111	0011011000100010	0011000000000011	0010011010000000	0101000011000010
	0001011011010000	1100000111100000	0011110000010001	0110001100001010	0101010100000100
	0001100100101100	1111000000010000	1000000000010010	1000101000000101	0001011100000001
	0111000010100000	0110001000100001	0001100011100001	0000010010001101	1001001010010100
	0010000000111001	1110101000000000	0000010110110000	1001000100000111	1010000001110000
	0000000111100100	1000001000101000	0000000010010001	0100100000110001	0000110010001010
	1010000000001101	1000001101011000	1010010000100100	0111000101000001	0101010000110010
	1001100000011001	0101001000101001	1001100000100010	0100010001000001	0001001010001010
	0000100101010100				
	10011010000101010	00001000101110101	10011000011000100	01011100100000101	01011010000110000
	10000011101000100	01001010011000110	11010100000100001	10110010010100010	01010100110000000
	10000010000100101	10110010000011000	01001100000011100	00001010100010001	01010000000000111
	00010001100010000	00011100010001011	00010001001011101	00100010100010110	01110001101001000
n =17	11100000001000110	00011010110001000	10001001001011000	01100000110001001	10000001010110001
	01001111000010001	00000100000000100	00100110110100001	10000100100001000	00101000101000000
102	00100100100001111	00010010101101000	01100101000011010	10101001000000010	00010001000101110
	11001100010000010	00010111100000110	01010101011100000	01101000000101001	10011110101000000
	11000011110001000	10000000110011011	00000100111101100	00000000101011110	10100100101000001
	01110000000111000	00000000010000010	01000110100100100	11000010000001110	01000000100100011
	01000010001010010	00001011010101100	00100001010101000	10001001000010011	10101110001100000
	00001000010111000	00100000001110010	00010010000110011	01000001010010110	11111000000000000
	01000000011000101	01010011000001000	00000010111010000	00111000000010100	00010100010011100
	11100110010000000	10010000101000111	10010110010010001	11000000000011001	10101000110000110
	00011001011010000	00100010010001011	00011100001000001	10000101000001101	11100001011001000
	00101001001100100	00110100010000101	10101000001001100	11010101000000100	00001011101001010
	01010000111010001	01001001100000010	00111000000100011	10000100001100010	00101100011010010
	01100000110110100	10000010100111000	00010001110101001	00011110000010010	11000111001010000
	00100111000001100	00000000001101001	01100101000100000	01100010001000001	10001001100100000
	00001110001001101	00001100000110110	01000100001001010	00110100001100100	00001101110000001
	01100011000000111	11001000110010000			
	000101010001110101	000000110100111100	000000101011101000	111010000101001001	
	010000101100100110	000101110001011000	001001000100100110	100010111000100000	101010001000101100
	100001101100000101	011001001000111000	101000001011010100	010000010111001001	100000011110110000
	000100011000010010	110001100001000000	000110011100000001	000111000110100010	000000000100011011
	000110000001101110	100001010001101100	011111010000000100	111100010000100010	011010100001000000
n =18	101110100011100000	010011001001001001	000001110010001001	100111000100000100	001010100100010101
	101000110001101000	100000110100010001	011010010000001110	100010110001000011	110000000001010101
126	000011100000100101	101100001100110001	100110010001100001	011001011010010000	010010010011110000
	010101000101000110	100001100010100100	001001011100001100	000001101000101110	011110000000101001
	000100110011000010	000011000101110000	011101000001001000	011100101010110000	001000001001110101
	000001000001000111	011001000011100001	010010011001010100	000110000000110111	100010001111000010
	100111000000100011	100000000111001100	000101000110001101	000100001100101000	001000000110110010
	000000101010000011	000001011101010001	001100001101000001	000001111111000000	101100010100010100
	010010100100011000	001101100010100001	001000010010100100	001111110110000000	010110100000001010
	100110010000011000	010000010001111010	011000100011011010	010000011110000100	100000001010001001
	000001101000010000	101001010000000101	110001010100110001	110101000000101100	010000010010000011
	111001001000000011	000100001011001110	001111001000000110	100010111101001000	001100110100000011
	010110000110000000	100001010100000010	001011111000000000	110100001010001100	000010100101000110
	100010010010000110	000010000010011100	000010010000000101	010000010100101101	011010010100100000
	000000000100000000	110001010010001000	001100101000001010	001000010001010000	000011001001100010
	100010101010010010	010000001000001000	111100001011000000	000000100111100011	000100100110010100
	001100000000011101	101010100110001000	010010001100010001	000011010010010010	100100100000110001
	011000100010000111	010001110000000110	001010001010010001	000100010101001100	010100000000110000
	101001000110001010	100001001001011010	101000100000110110	010100101000000001	100000001000110111
	010000001010011110	001011000000011010	001000000011001011	000010100100101011	110000100100101000
	100101000010000000	000011010000101000			

	0100000101100001100	0010000000001100111	0001111001001010000	1001010001000100101	1001101000000010010
	0000110000000110111	1000010000010001101	1101000010101100000	0101000110100000011	1000001011001100000
	0001001101000110110	0100010010000010111	0100010101000000011	1111001001010000000	0110000100001010011
***n* =19**	1010000001101000010	1011010000100100010	0000001010101001011	1001000100110000010	0101010000001000100
	0001010000011110010	0100101000000000000	0101101000001110001	1001100001010110000	0100101000100000111
159	0001001001111000101	0010100010011000010	0000100101011100100	0000100010100110100	0101000010010011010
	0010001001011100001	0000001100010100010	1110010110000000000	1100001000010000001	0001001110010001000
	0010100100010001101	0000101001000101100	0011100000001001101	0000010100011101010	1011100100101010000
	0100101111000000010	0001010011010000100	0110010001001100010	0000010011011010001	0010010000110110100
	0000000101110101010	1001001101000001100	0100000111001111000	0110001001000000110	1001010001000011001
	0100010100010100101	1100100100110001000	0010010001011001100	0100011000100101001	0100001011000010001
	1010010101100000001	0111000000100001001	0010100011110100010	1010110000001000110	0100001110111100100
	0001010000111001001	0110100001111000000	1001100110010000100	1110100101000000000	1001011010000010011
	0011101100100000011	1010110001000001100	1010001100001000000	0001000110000101101	0110100110001000001
	0001000000000010110	0001110010101000000	1000110100000010010	0100100001000001011	1000111010010000000
	1011100011001010000	0110100101000010101	1000000001101011100	1010010000010110001	0010010100111000010
	0010110000100100001	0011000100000101110	0010001000110000100	0000001001100110000	0001111010000010100
	0010111000000011001	0001010110010000011	1100011001100000010	0000011010000001100	1000100001100100011
	0000010101001010100	1110000000100010100	0010101010101001100	0000110000100001010	0111100100100100000
	1010000011000000101	0100000100101011110	0010011011100100101	1000000000000011011	1100100000100100100
	0010101100010010000	0001100001110001000	0100010100100111000	0100011110010010000	1100001100001001100
	1100000101010010100	1000000010110010011	0000010110100011010	0000011010100100010	0101100010010000001
	1001000011100001110	0010001010001010101	0100100000101101011	0100000000011011101	0000100000001011000
	1010000110010100010	1000010000101010001	0011010100000010000	0100000011110100100	1101000000011101000
	0000111100001000101	1000011001010011010	1001100100000001001	1101010000000001110	1011101000000100001
	0110010010010101001	0110001010000111000	1000000011000110010	0110111100000100100	0000000100101100000
	0100110010010000110	0100111100101010000	0001100001010000111	0001101110001100000	0000001101010001101
	0001010000000101000	1000010101000101110	0001000100110010101	1101001100001000011	1000000010100101001
	1000001010010101110	1101000100000110001	0011000001001111000	0100100000110010010	0010000110010010110
	1000001000111110000	0100000010111000011	1111100010000000100	0100110001001001000	0010001000010001011
	0001110101000100000	1001000001001100110	1110010000001001001	1000100110101000010	0011001011000000000
	0000011101110000000	0111000000010100100	0010010011000011000	0011000000110011010	
	00100010000111010110	01110000110000110000	00110010001000100000	10000000001011010011	00001011000110110010
	10000011001100100101	00000110000100100110	00100000101100100011	11100110000100001010	00010111001010010001
	00000101101000101101	00000000100101110001	00100010100010001000	00010001010101001101	00101101000100010110
***n* =20**	00100110010010001101	01000101100000001010	11000000100110100010	10001110011001000001	00000110100001110000
	10000010010001010110	00000010110000111100	01010001100001000100	11100100000000101001	01010000110100011100
	10110100101000010100	11001000110001010000	01110011010000001001	00000100000010100001	00110000011101000010
	01010000011010101100	00100100101011101000	01000100000100111010	01010100000010110100	01000010011101010100
195	01010011101010000000	00000101001010111110	00010100001101110000	01000011000000111011	11000110100100010000
	00000111010101001010	00010010100000010110	11000011001010011000	01101000010001000010	10010000001100000110
	00001010010010010001	00010100000000001100	10100100010001110001	10010010100010010001	00110001010000000111
	00000001110010101000	01100000011000000101	10000010101011000000	10001001000100110000	00000001001101101010
	10101000100000000111	01010010000110001010	01000111010000100100	00000001111110010101	00100110001110110000
	00101100010101000101	10010110000001100011	01100000001111001000	00111001001100001100	01011101010100000001
	00101011010100000000	00111101000011000010	10000100011100001001	00011010011011000000	01011100001001001000
	10001000000111100101	10100001000000011001	00010010100110101001	00001000011000001100	00000000000000000010
	00000101010000110011	00011010000010100100	11010110110000000001	10100011000110100001	01000001001010000111
	10111001011000000001	01000001011000011000	10011000000000000001	00101000011000110011	10101010011101100000
	10000010010100101000	11000001010100000100	00011001100100000011	10000010001010001101	11001100010101010010
	11010100001000010011	10100010011000010000	10010001000001001011	01001001000101000000	00011111000001100000
	10000100000111001000	00101011001000000011	10011010000100011010	11101101000010000001	01000101000111010001
	00100001100110011100	11110101010000000010	00101101010000111000	00111100100000110011	01011010001000110000
	00010000000100111101	00010100011100100101	00011000111001010100	11010001001000101000	00010100000001011011
	00000010110100001111	01000010010011101010	11010000010111000001	10101000101100000000	01000100100100001101
	10010100011010000010	11000000101000110101	10100110000000000100	00001011111000001000	01000011110100000001
	11010100100001100100	01101110100010100000	00011000101110100000	00001100100011000011	11000010000101001001
	01101011100001001000	11011100000110001100	00001001110001100101	00001010000101000111	00110000001010011010
	00001100101000010110	10011011000001010100	01000000000001011100	00101100000100011001	10100100110010001010
	00100101110001000001	10000010101000110010	00110100010011001000	11000000111110001000	11110010000010000101
	10100000000010111100	10111000100001001000	00010000010110010000	00010001101101000001	10000100100001001110
	01100001001100000010	00100100000001101110	10000000110001101001	00011100000010101010	01010000000001100111
	10011000001110011001	00000101111101010000	00001110011100000010	00000001011011000010	00010100101111000100
	11000010001001100110	01001000001110010010	01101001100000100100	00101000110010010100	01000010101000001100
	00001001000011011000	00001000100101011010	00100010010001011011	01000000111001100000	10101000000010100010
	01001001000100101101	01001101001000100010	00001110100100001100	11001000010000001011	00110111000000010101
	10100001101010001011	00010101100010010010	10110001100101010000	00100100001100011100	01100011110000010010
	01110000100011010001	00010110010000101001	10000001100000100110	01100010000000110101	11001000101100000000
	00001000011011111000	00011010110000100011	01110100101010100010	10001100110000011001	11000001011001000001
	11011001101000000000	11100001000011010000	00111010000101010000	00100001001001110101	01001000101010001001
	10010100010101010100	00010010001001001001	10000101100110000100	00001101010111100000	01100110001001000001